LABORATORY FOR COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

(4)

MIT/LCS/TR-435

SIMULATING $(\log^c n)$-wise INDEPENDENCE IN NC

Bonnie Berger
John Rompel

May 1989

545 TECHNOLOGY SQUARE CAMBRIDGE, MASSACHUSETTS 02139

(4)

MIT/LCS/TR-435

SIMULATING $(\log^c n)$-wise INDEPENDENCE IN NC

Bonnie Berger
John Rompel

May 1989

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) MIT/LCS/TR-435	5. MONITORING ORGANIZATION REPORT NUMBER(S) N00014-80-C-0622
6a. NAME OF PERFORMING ORGANIZATION MIT Laboratory for Computer Science 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research/Department of Navy
6c. ADDRESS (City, State, and ZIP Code) 545 Technology Square Cambridge, MA 02139	7b. ADDRESS (City, State, and ZIP Code) Information Systems Program Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA/DOD 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Boulevard Arlington, VA 22217	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) Simulating (log^c n)-wise Independence in NC	
12. PERSONAL AUTHOR(S) Berger, B. and Rompel, J.	
13a. TYPE OF REPORT Technical 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Year, Month, Day) May 1989 15. PAGE COUNT 15
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD / GROUP / SUB-GROUP	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Parallel algorithms, NC, randomness, discrepancy, edge coloring, hypergraph coloring

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

We develop a general framework for removing randomness from randomized NC algorithms whose analysis uses only polylogarithmic independence. Previously no techniques were known to determinize those RNC algorithms depending on more than constant independence. One application of our techniques is an NC algorithm for the set discrepancy problem, which can be used to obtain many other NC algorithms, including a better NC edge coloring algorithm. As another application of our techniques, we provide an NC algorithm for the hypergraph coloring problem.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Judy Little Publications Coordinator	22b. TELEPHONE (Include Area Code) (617) 253-5894 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

☆U.S. Government Printing Office: 1985—607-047

Unclassified

Simulating $(\log^c n)$-wise Independence in NC

Bonnie Berger*
John Rompel†

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

We develop a general framework for removing randomness from randomized NC algorithms whose analysis uses only polylogarithmic independence. Previously no techniques were known to determinize those RNC algorithms depending on more than constant independence. One application of our techniques is an NC algorithm for the set discrepancy problem, which can be used to obtain many other NC algorithms, including a better NC edge coloring algorithm. As another application of our techniques, we provide an NC algorithm for the hypergraph coloring problem.

Keywords: Parallel algorithms, NC, randomness, discrepancy, edge coloring, hypergraph coloring

*supported in part by Air Force Grant AFOSR-86-0078

†supported by a National Science Foundation Graduate Fellowship, DARPA contract N00014-80-C-0622, and Air Force AFSOR-86-0078

1 Introduction

A fundamental issue of theoretical computer science is the degree to which randomness helps in computation. In many cases, the most natural algorithm to solve a problem involves randomness. Often, however, it is possible to convert a randomized algorithm into a deterministic one.

For many applications [KW,L1,ABI], the problem of removing randomness from an algorithm can be solved by finding an $X =< X_1, \ldots, X_n >$ such that $F(X) \geq E[F(X)]$, for some function F and some sample space $\mathcal{S}$ over which the expectation is to be computed. The problem is then how best to find a good sample point (e.g., an X such that $F(X) \geq E[F(X)]$) in $\mathcal{S}$. If the space of sample points is small (e.g., polynomial), then this can be accomplished by brute force; namely, we could try all points until we get a good one, for one must exist. However, typical sample spaces are larger than polynomial, and brute force is too expensive. In such situations, the only general method available is due to Raghavan and Spencer [Rag,S]. Their method works by setting the X_i's one by one in such a way as to not decrease the conditional expectation (e.g., setting X_{i+1} so that $E[F(X) \mid X_1, \ldots, X_{i+1}] \geq E[F(X) \mid X_1, \ldots, X_i]$). The main difficulty with this approach is computing the conditional expectations — the ability to do so determines when the method can and cannot be used.

Unfortunately, the Spencer/Raghavan method is inherently sequential. Moreover, the running time is at least n. Since the best time one could hope for is logarithmic in the size of the sample space, for large probability spaces this is probably as good as one can get; yet, for smaller spaces, n is far from optimal. The only improvement to this approach was by Luby [L2] who showed how to search in time logarithmic in the size of the sample space for the special case of pairwise independence, thereby improving the processor efficiency of several NC algorithms ($\Delta + 1$ vertex coloring, MIS, and maximal matching) from $n^2(n+m)$ to $n+m$.

In this paper, we show how to search in time logarithmic in the size of the sample space for a wide range of functions F and arbitrarily large sample spaces. As a result, we can prove substantially stronger results than is possible with the Luby method. In particular, we are able to derive NC algorithms for several problems that were not previously known to be in NC, and we can search $(\log^c n)$-wise independent $n^{\log^c n}$-size sample spaces in NC.

In Section 2, we demonstrate how our techniques apply to the problem of set discrepancy. In this problem, we are given a set of n points and n subsets of at most Δ of these points, and we want to color the points 0 and 1 so that the discrepancy, or maximum difference between the number of 0's and the number of 1's in any subset, is small. The best known randomized (parallel) algorithm achieves a discrepancy of $\sqrt{\Delta \log n}$; Raghavan and Spencer applied their technique to this algorithm to obtain a deterministic sequential algorithm with the same bound. We show that the randomized algorithm requires $\frac{\log n}{\epsilon \log \Delta}$-independence to give a discrepancy bound of $\Delta^{1/2+\epsilon}\sqrt{\log n}$ for any fixed $\epsilon > 0$; then we apply our techniques to convert this to an NC algorithm which attains the same bound (using $n^{1/\epsilon}$ processors). This algorithm has many applications. As an example, we give a deterministic version of the Karloff-Shmoys parallel edge coloring algorithm [KS], obtaining the same $\Delta + \Delta^{1/2+\epsilon}$ bound on the number of colors used as their randomized algorithm and beating the known deterministic bound of $2\Delta - 1$ [L2].

In Section 3, we show how to apply our techniques to a large class of problems which depend on $(\log^c n)$-wise independence. In particular, we describe an NC algorithm for obtaining the expected value of any function which is the sum of a polynomial number of terms, each depending on $O(\log n)$ binary random variables; e.g., a function of the form

$$F(X) = \sum_{i=1}^{n^a} f_i(X_{i,1}, \ldots, X_{i,b\log n}).$$

Alternatively, we can allow the terms to be simple functions of $\log^c n$ random variables; e.g., characteristic functions of affine subspaces, which by a reduction can be used to build any function which is non-zero for only a polynomial number of points.

Finally, in Section 4, we give several methods for extending our technique to multi-valued random variables. As an illustration, we consider the hypergraph coloring problem: given a d-uniform hypergraph $(V, \mathcal{E})$, color the vertices with d colors so that at least $d!|\mathcal{E}|/d^d$ edges have one vertex of each color. If d is a constant, this problem can be solved by trying all sample points in a d-wise independent distribution [ABI]. Using our techniques for handling $(\log n)$-wise independent multi-valued random variables, we give a deterministic NC algorithm which solves this problem for all d. The particular technique used to handle the multi-valued random variables in this problem involves generating and solving a series of problems with binary random variables, highlighting the importance of being able to solve a large class of these.

Our generalization of Luby's work to remove randomness from $(\log n)$-wise independent discrepancy-based problems such as edge coloring was worked out in November, 1988, and first appears in [B]. Our general framework for removing randomness from $(\log^c n)$-wise independent NC calculations was worked out and written up by mid-December, 1988. In January, 1989, Motwani, Naor, and Naor [MNN] obtained our results for discrepancy-based problems such as edge coloring.

2 An Example of the Method — Set Discrepancy

2.1 Definition of Problem

Spencer [S, p. 30] defines the *set discrepancy problem* as follows. Let $\mathcal{A} \subseteq 2^\Gamma$, $|\mathcal{A}| = |\Gamma| = n$, be a family of finite sets. Let $\chi : \Gamma \to \{-1, +1\}$ be a 2-coloring of the underlying points. Define

$$\begin{aligned} \chi(A) &= \sum_{i \in A} \chi(i) \\ disc(\chi) &= \max_{A \in \mathcal{A}} |\chi(A)|. \end{aligned}$$

We want to find a χ such that $disc(\chi)$ is small.

How small can we make $disc(\chi)$? Spencer [S, p. 73-77] shows that there exists a χ with $disc(\chi) = O(\sqrt{n})$. He also shows that this is the best possible; i.e., that there exists an $\mathcal{A}$ such that all χ have $disc(\chi) = \Omega(\sqrt{n})$.

It is interesting to bound discrepancy in terms of maximum degree $\Delta = \max_{A \in \mathcal{A}} |A|$. Spencer's lower bound can be easily modified to give, for any Δ, an $\mathcal{A}$ with cardinality n and maximum degree Δ such that all χ have $disc(\chi) = \Omega(\sqrt{\Delta})$. It is easily shown in Section 2.2 that if we pick χ at random, with high probability $disc(\chi)$ will be at most $2\sqrt{\Delta \lg n}$. This immediately gives an RNC algorithm achieving that bound. Raghavan and Spencer [Rag,S] show how to convert this into a deterministic poly-time algorithm. In the sections to follow, we develop an NC algorithm which outputs a χ with $disc(\chi) \leq \Delta^{1/2+\epsilon}\sqrt{\lg n}$.

An interesting special case of the set discrepancy problem is the *graph discrepancy problem*. Given a graph $G = (V, E)$, we want to find a 2-coloring of the vertices $\chi : V \to \{-1, +1\}$ such that $\max_{v \in V} |\sum_{u \in N(v)} \chi(u)|$ is small. We can reduce this problem to set discrepancy by letting $\Gamma = V$ and $\mathcal{A} = \{N(v) \mid v \in V\}$. Plugging in our NC algorithm for discrepancy, we will get a χ with $\max_{v \in V} |\sum_{u \in N(v)} \chi(u)| \leq \Delta^{1/2+\epsilon}\sqrt{\log n}$, where Δ is the maximum degree of G.

2.2 An RNC Algorithm

Consider the following algorithm for set discrepancy: randomly pick χ until $disc(\chi) \leq 2\sqrt{\Delta \ln n}$. One iteration of this is clearly in RNC. We will show that the expected number of iterations is less than two. The following lemmas will prove useful.

Lemma 2.1 (S, p. 29) *Let $X_1, \ldots, X_\Delta$ be independent and identically distributed with $Pr[X_i = +1] = Pr[X_i = -1] = 1/2$. Let $S = \sum_i X_i$. Then $Pr[S > \lambda] < e^{-\lambda^2/2\Delta}$.*

Lemma 2.2 $Pr\left[disc(\chi) > 2\sqrt{\Delta \ln n}\right] < 2/n$.

Proof Straightforward application of Lemma 2.1. □

Thus, the expected number of iterations is at most $1/\left(1 - \frac{2}{n}\right) < 2$. So the above is clearly an RNC algorithm. Also, one can easily show using Lemma 2.2 that $E[disc(\chi)] = 2\sqrt{\Delta \ln n}$.

2.3 The Overall Approach

Lemma 2.2 shows that the probability of $disc(\chi)$ being larger than $2\sqrt{\Delta \ln n}$ is small. This implies that there exists a χ with $disc(\chi) \leq 2\sqrt{\Delta \ln n}$. We wish to find such a χ deterministically. Unfortunately, the random construction of Section 2.2 assumed a fully independent distribution, which must have 2^n sample points. Clearly, we cannot search this sample space exhaustively. However, Raghavan and Spencer [R,S] developed a method to perform a binary search on the sample space. While they achieve a polynomial time algorithm, it requires n decisions to be made. Since each decision depends on previous ones, it seems very unlikely that these decisions could be made in parallel. To get an efficient parallel algorithm, we must work with a smaller distribution space. A natural choice is a k-wise independent distribution, where k is small.

Definition 2.3 *$X_1, \ldots, X_n$ are k-wise independent if for any k-subset of $X_1, \ldots, X_n$ and for any $r_1, \ldots, r_k$,*

$$\begin{aligned} &Pr[X_{i_1} = r_1 \wedge X_{i_2} = r_2 \wedge \cdots \wedge X_{i_k} = r_k] \\ &= \quad Pr[X_{i_1} = r_1] Pr[X_{i_2} = r_2] \cdots Pr[X_{i_k} = r_k]. \end{aligned}$$

Ideally our goal is to find a χ with small discrepancy by finding a χ for which $disc(\chi) \leq E[disc(\chi)]$, where the expectation is taken over a k-wise independent distribution for some small k. The choice of k is influenced by two factors:

1. if k is too small, then $E[disc(\chi)]$ might be too large and
2. if k is too large, then finding a χ which achieves the expectation takes too long.

As a compromise, we will eventually choose $k = \frac{\log n}{\epsilon \log \Delta}$, $\epsilon > 0$.

There are other problems with this approach, however. Most important, to find a good χ, we will need to compute expectations of $disc(\chi)$ conditioned on some knowledge of the distribution in NC. This is hopelessly complicated by the *max* and absolute value in $disc(\chi)$. To get around these problems, we will use higher moments, and use $\sum_{A \in \mathcal{A}} |\chi(A)|^k$ as an upper bound on $disc^k(\chi)$. For even k, this gets rid of both the max and the absolute value. In other words, we will

1. show that $E\left[\sum_{A \in \mathcal{A}} |\chi(A)|^k\right]$ is small for suitable k where the expectation is taken over a k-wise independent distribution, and then

2. find a χ such that $\sum_{A\in\mathcal{A}} |\chi(A)|^k \leq E\left[\sum_{A\in\mathcal{A}} |\chi(A)|^k\right]$.

As a consequence, we will have found a χ for which $disc^k(\chi)$ is small, and thus for which $disc(\chi)$ is small. By choosing $k = \frac{\log n}{\epsilon \log \Delta}$, we will produce a χ for which $disc(\chi) \leq \Delta^{1/2+\epsilon}\sqrt{\log n}$. This is not quite the $\sqrt{\Delta \log n}$ bound we got with the RNC algorithm, but it is close.

2.4 Bounding the Independence Needed

Our first task is to bound $E\left[\sum_{A\in\mathcal{A}} \chi^k(A)\right]$. This is accomplished in the following lemmas.

Lemma 2.4 *Any function which is the sum of functions depending on at most k random variables each has the same expected value taken over any distribution with k or greater independence.*

Proof Follows directly from the definition of k-wise independence and linearity of expected value. □

Lemma 2.5 *For any even k-wise independent distribution and for all $A \in \mathcal{A}$,*

$$E[\chi^k(A)] \leq O((k\Delta/e)^{k/2}).$$

Proof $E[\chi^k(A)] = 2\sum_{i=1}^{\infty} i^k Pr[\chi(A) = i]$. Break up the sum into blocks of $\sqrt{\Delta}$. For $i \in \left[1,\ldots,\sqrt{\Delta}\right]$, i^k is bounded above by $\left(\sqrt{\Delta}\right)^k$; for $i \in \left[\sqrt{\Delta}+1,\ldots,2\sqrt{\Delta}\right]$, i^k is bounded above by $\left(2\sqrt{\Delta}\right)^k$, and so on. Then

$$\begin{aligned} 2\sum_{i=1}^{\infty} i^k Pr[\chi(A) = i] &\leq 2\sum_{r=1}^{\infty} (r\sqrt{\Delta})^k Pr[(r-1)\sqrt{\Delta} < \chi(A) \leq r\sqrt{\Delta}] \\ &< 2\sum_{r=1}^{\infty} (r\sqrt{\Delta})^k e^{-(r-1)^2/2} \end{aligned}$$

(where the last inequality is from Lemma 2.1). The terms of the sum geometrically increase to a point, and then at some large r, they begin geometrically decreasing. Use the ratio test to get a sense of this:

$$\frac{(r+1)\text{st term}}{r\text{th term}} = \frac{\left(1+\frac{1}{r}\right)^k}{e^{(2r-1)/2}} = e^{k/r-\theta(k/r^2)-r+1/2}.$$

So for small r the terms are growing at least geometrically, and for large r they decrease at least geometrically. The maximum term is about where $e^{k/r-\theta(k/r^2)-r+1/2} \approx 1 = e^0$, which occurs at $r = \sqrt{k} + \theta(1)$. The constant will go away in the O notation, so we will ignore it. We could show that this ratio will be bigger than e or less than $1/e$ more than one or two terms away from $r = \sqrt{k}$. Hence,

$$E[\chi^k(A)] = O\left(\left(\sqrt{k}\sqrt{\Delta}\right)^k e^{-k/2}\right) = O\left(\left(\frac{k\Delta}{e}\right)^{k/2}\right).$$

The above calculations assume a fully independent (or at least Δ-wise independent) distribution. However,

$$\chi^k(A) = (\sum_{i\in A} \chi(i))^k = \sum_{i_1\in A}\sum_{i_2\in A}\cdots\sum_{i_k\in A} \chi(i_1)\cdots\chi(i_k).$$

So Lemma 2.4 applies to show that any k-wise independent distribution gives the same expected value. □

Corollary 2.6 *For any even k-wise independent distribution,*

$$E[\sum_{A\in\mathcal{A}} \chi^k(A)] \leq n\,O((k\Delta/e)^{k/2}).$$

We can now give a lower bound on the value for k. We want $E[\sum_{A\in\mathcal{A}} \chi^k(A)]^{1/k} \leq \Delta^{1/2+\epsilon}\sqrt{\log n}$; this is roughly captured by having $n^{1/k} \leq \Delta^{\epsilon}$. This implies we need $k = 2\left\lceil \frac{\log n}{2\epsilon \log \Delta} \right\rceil$. If k is thus, and we are able to find a χ such that $\sum_{A\in\mathcal{A}} \chi^k(A)$ is at most its expectation, this implies that

$$disc^k(\chi) \leq \sum_{A\in\mathcal{A}} \chi^k(A) \leq E[\sum_{A\in\mathcal{A}} \chi^k(A)] \leq O(n(k\Delta/e)^{k/2}).$$

So,

$$\begin{aligned} disc(\chi) &\leq O(n^{1/k}\sqrt{k\Delta/e}) \\ &\leq O(\Delta^{\epsilon}\sqrt{k}\,\sqrt{\Delta}) \\ &\leq \Delta^{1/2+\epsilon}\sqrt{\log n}. \end{aligned}$$

Remark 2.7 *For any n and Δ, we can construct a $\left(\frac{2\log n}{\log \Delta}\right)$-wise independent distribution and a set system $\mathcal{A}$ with maximum degree Δ such that $E[disc(\chi)] = \Omega(\Delta)$. Therefore, to get $disc(\chi) \leq \Delta^{1/2+\epsilon}$, any method based on independence alone will require $\left(\frac{2\log n}{\log \Delta}\right)$-wise independence.*

The next three sections will be devoted to finding a χ such that $\sum_{A\in\mathcal{A}} \chi^k(A)$ is at most its expectation. If we let $\chi(i) = (-1)^{X_i}$, finding a χ is identical to finding an X which satisfies this relation. From this point on, we will let

$$F(X) = -\sum_{A\in\mathcal{A}} \chi^k(A) = -\sum_{A\in\mathcal{A}} \sum_{i_1\in A} \cdots \sum_{i_k\in A} (-1)^{X_{i_1}+\cdots+X_{i_k}}.$$

So henceforth we want an X such that $F(X) \geq E[F(X)]$.

2.5 Generating k-Wise Independent Variables

It still remains to demonstrate a k-wise independent distribution on which we can perform a binary search efficiently in parallel.

Luby [L2] gave the following such probability space for the case $k = 2$. Let $l = \lceil \log n \rceil + 1$ and ω be picked uniformly from Z_2^l. For each point i, let $<i_1, \ldots, i_{l-1}>$ be the binary expansion of i. Define random variables $X_1, \ldots, X_n$ so that

$$X_i = \left(\sum_{j=1}^{l-1} (i_j\omega_j) + \omega_l\right) \bmod 2.$$

Observe that Luby's distribution is not 4-wise independent since, in particular, X_4, X_5, X_6, and X_7 are dependent.

We extend Luby's distribution to be k-wise independent for all k as follows. We will assign a label $a_i \in Z_2^l$ to each point i, where l is bounded by some polylogarithmic function of n. Pick $\omega \in Z_2^l$ uniformly at random, and let

$$X_i = a_i \cdot \omega.$$

Note that we can express Luby's distribution in this framework by leting $a_i = <i_1, \ldots, i_{l-1}, 1>$.

The main benefit of our distribution is that we can now give a necessary and sufficient condition for the X_i's to be independent and unbiased. The following result is similar to others in the literature [MS,ABI] and can probably be considered to be well-known.

Theorem 2.8 *$X_{i_1}, \ldots, X_{i_k}$ are independent and unbiased if and only if $a_{i_1}, \ldots, a_{i_k}$ are linearly independent as vectors over Z_2.*

Proof Utilizes elementary linear algebra. □

To get $X_1, \ldots, X_n$ which are $(\log n)$-wise independent, we need a set of labels $a_1, \ldots, a_n$ such that every $\log n$ of them are linearly independent. (By Theorem 2.8, this gives us $\log n$-wise independence of the X_i's.) In fact, it suffices to get an $n \times r$ matrix over $GF(2^s)$ with the property that any $\log n$ rows are linearly independent. Letting a_i be the ith row with each element $\alpha_0 + \alpha_1 x + \cdots + \alpha_{s-1} x^{s-1} \in GF(2^s)$ expanded out to $< \alpha_0, \ldots, \alpha_{s-1} >$ gives length $l = rs$ labels such that any $\log n$ are linearly independent over Z_2. Several well-known ways of getting such matrices, for $l = O(\log^2 n)$, are described in [ABI, Rab]. Even randomly chosen labels of this length will work.

Theorem 2.9 *For a random set of labels $\{a_1, \ldots, a_n\} \subseteq Z_2^{2\log^2 n}$, $X_1, \ldots, X_n$ are $(\log n)$-wise independent with high probability.*

Proof Omitted. □

Since any probability space with n k-wise independent random variables must contain $\Omega((n/k)^{\lfloor k/2 \rfloor})$ sample points [ABI,CGHFRS], the labels $a_1, \ldots, a_n$ must be $\Omega(\log^2 n)$ bits long.

One final thing to note is that we can easily extend the above to give $(\log^c n)$-wise independence for any c by using the same techniques to obtain labels with $l = O(\log^{c+1} n)$, also matching the lower bound.

2.6 Zeroing in on a Good Sample Point

Now that we have a k-wise independent distribution, we will explain how to do a binary search on it efficiently in parallel. We will use the strategy introduced by Luby in [L2] for zeroing in on a "good" sample point; i.e., an ω such that $F(X) \geq E[F(X)]$.

To zero in on a good ω, one bit of ω is determined at a time, thereby performing a binary search on the ω's. This is done as follows. At the beginning of iteration t, assume we have set $\omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}$. Then we compute $E[F(X) \mid \omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}, \omega_t = 0]$ and $E[F(X) \mid \omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}, \omega_t = 1]$. We then set ω_t to the $s_t \in \{0, 1\}$ which maximizes $E[F(X) \mid \omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}, \omega_t = s_t]$. We will show how to compute these conditional expectations in Section 2.7.

Lemma 2.10 *After step t of the above procedure, $E[F(X) \mid \omega_1 = s_1, \ldots, \omega_t = s_t] \geq E[F(X)]$.*

Proof (by induction on $|s|$)
The case $|s| = 0$ is clearly true. Assume this lemma is true for $t - 1$; i.e., we have $E[F(X) \mid \omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}] \geq E[F(X)]$. Then

$$
\begin{aligned}
& E[F(X) \mid \omega_1 = s_1, \ldots, \omega_t = s_t] \\
&= \max(E[F(X) \mid \omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}, \omega_t = 0], E[F(X) \mid \omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}, \omega_t = 1]) \\
&\geq (E[F(X) \mid \omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}, \omega_t = 0], E[F(X) \mid \omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}, \omega_t = 1])/2 \\
&= E[F(X) \mid \omega_1 = s_1, \ldots, \omega_{t-1} = s_{t-1}] \\
&\geq E[F(X)] \quad \text{(by inductive hypothesis).}
\end{aligned}
$$
□

Corollary 2.11 *The output of the above procedure is an X such that $F(X) \geq E[F(X)]$.*

2.7 Computing Conditional Expectations

In general, computing conditional expectations is hard to do and separates when one can and cannot use Raghavan/Spencer-like techniques to zero in on a a good sample point. Fortunately, in the case of discrepancy, we have devised a simple and efficient approach for computing conditional expectations. Recall that to solve discrepancy, we need to compute conditional expectations $E[F(X) \mid \omega_1 = s_1, \ldots, \omega_t = s_t]$ where $F(X)$ is of the special form

$$F(X) = -\sum_{A \in \mathcal{A}} \sum_{i_1 \in A} \cdots \sum_{i_k \in A} (-1)^{X_{i_1} + \cdots + X_{i_k}}.$$

Using linearity of expected value, we can break this up into components

$$\begin{aligned} h_{i_1 \ldots i_k}(s) &= E[(-1)^{\sum_{j=1}^{k} X_{i_j}} \mid \omega_1 = s_1, \ldots, \omega_t = s_t] \\ &= E[(-1)^{\sum_{j=1}^{k} a_{i_j} \cdot \omega} \mid \omega_1 = s_1, \ldots, \omega_t = s_t] \\ &= E[(-1)^{\bar{a} \cdot \omega} \mid \omega_1 = s_1, \ldots, \omega_t = s_t] \end{aligned}$$

where $\bar{a} = \sum_{j=1}^{k} a_{i_j}$. Let r be the last position which contains a 1 in $\bar{a}$. If $t < r$, then $\bar{a} \cdot \omega$ is unbiased, and therefore $h_{i_1 \ldots i_k}(s) = 0$. Otherwise, $\bar{a} \cdot \omega$ is the same for all ω which extend s, and hence $h_{i_1 \ldots i_k} = E[(-1)^{\bar{a} \cdot \omega}]$. Assuming we have precomputed $\bar{a}$ and r, we can compute $h_{i_1 \ldots i_k}(s)$ in constant time by extending a partial sum $\sum_{j=1}^{t} a_j s_j$ at each iteration and outputting $h_{i_1 \ldots i_k}(s) = 0$ if $t < r$ and $h_{i_1 \ldots i_k}(s) = (-1)^{\sum_{j=1}^{t} a_j s_j}$ if $t \geq r$.

To compute $E[F(X) \mid \omega_1 = s_1, \ldots, \omega_t = s_t]$, we need one processor for each possible $< A, i_1, \ldots, i_k >$, that is, at most $n\Delta^k$ total. Therefore, k must be $O(\frac{\log n}{\log \Delta})$. Letting k be the minimum possible, $2\lceil \frac{\log n}{2\epsilon \log \Delta} \rceil$, implies that $n^{3+1/\epsilon}$ processors is sufficient.

Then we can compute all $h_{i_1 \ldots i_k}(s)$ terms in parallel in constant time and sum them to get $E[F(X) \mid \omega_1 = s_1, \ldots, \omega_t = s_t]$ in $O(\log n)$ time. Thus we spend $O(l \log n)$ time in the l iterations of our procedure. In addition, we can perform the precomputation required above in $O(l \log n)$ time as well. Since $l = O(\log^2 n)$, this yields an $O(\log^3 n)$ algorithm for discrepancy.

2.8 Application to Edge Coloring

An *edge coloring* of a graph $G = (V, E)$ is an assignment of colors to all edges of the graph, so that any two edges that share a common vertex are assigned different colors. Let Δ be the maximum degree in G. Observe that any coloring requires at least Δ colors. In fact, Vizing [V] implicitly gives a polynomial time algorithm to $\Delta + 1$ color any graph. Karloff and Shmoys [KS] provide a parallel implementation of this algorithm to get a $\Delta + 1$ coloring of any graph in time $O(\Delta^{O(1)} \log^{O(1)} n)$ using a polynomial number of processors. Also of interest, there exist NC algorithms for optimally coloring bipartite graphs with Δ colors [GK,LPV,CH,AIS]. Furthermore, there is a trivial NC algorithm to $2\Delta - 1$ color any graph by $\Delta + 1$ vertex coloring [L2] the line graph. Berger and Shor [BS], Karloff [K], and Naor [N] found NC algorithms to $\Delta + \Delta/\log^{O(1)} n$ color any graph.

Of particular interest here, there is an RNC algorithm in [KS] which $\Delta + \Delta^{1/2+\epsilon}$ colors any graph. We will remove the randomness from this algorithm by using the techniques discussed above. The RNC algorithm, *Algorithm A*, is as follows:

1. If $\Delta < (\log n)^{1/\epsilon}$, then use the [KS] $\Delta + 1$ deterministic algorithm.

2. Run an RNC algorithm for graph discrepancy, randomly picking χ until $disc(\chi) \leq \Delta^{1/2+\epsilon}$. Let $A = \{v \mid \chi(v) = +1\}$ and $B = \{v \mid \chi(v) = -1\}$. This partition gives us two graphs, both with vertex set V: bipartite graph G_1, which has all the edges of G between A and B; graph G_2 which has all the other edges of G.

3. Color G_1 using the Δ coloring algorithm for bipartite graphs.

4. Recurse on G_2, using a new set of colors.

This algorithm works because the above partition implies that both G_1 and G_2 have maximum degree at most $\Delta/2 + \Delta^{1/2+\epsilon}$.

To make *Algorithm A* deterministic, we need only demonstrate a deterministic method for graph discrepancy, which we said in Section 2.1 was a special case of the set discrepancy problem. Plugging in the set discrepancy results with $\epsilon' = \epsilon/2$, we get a χ such that $disc(\chi) \leq \Delta^{1/2+\epsilon'}\sqrt{\log n}$. Note that $\Delta \geq (\log n)^{1/\epsilon}$, since we handled the other case in Step 1 of *Algorithm A*. Thus, $\sqrt{\log n} \leq \Delta^{\epsilon/2}$. So we have $disc(\chi) \leq \Delta^{1/2+\epsilon}$, which implies $\chi(N(v)) \leq \Delta^{1/2+\epsilon}$ for all $v \in V$.

3 Setting up a General Framework

For discrepancy-based problems, we considered a very specific class of functions, namely those of the form $\sum_{i=1}^{n^a}(-1)^{\sum_{j=1}^{k} X_{i,j}}$, and showed how to achieve the expected value for these. What can we do in general? In particular, for which functions can we compute conditional expectations (the method of Section 2.6 will then apply to achieve the expected value)? In order to give ourselves a fighting chance, we will restrict our attention to functions of the form

$$F(X) = \sum_{i=1}^{m} f_i(X_{i,1}, X_{i,2}, \ldots, X_{i,k}).$$

These are exactly the functions for which we can apply Lemma 2.4 to show that k-wise independence gives the same expected value as full independence. Since we will require at least one processor for each f_i term, we will require m to be polynomial in n. In Section 3.1, we will show how to compute conditional expectations for arbitrary[1] f_i when $k = O(\log n)$. In Section 3.2, we will describe the f_i's for which we can handle the case $k = O(\log^c n)$.

3.1 Logarithmic Number of 0/1 Variables

In this section, we will present two different methods for computing conditional expectations for functions of the form

$$F(X) = \sum_{i=1}^{n^a} f_i(X_{i,1}, \ldots, X_{i,b\log n}).$$

We present both methods because, depending on what problem we wish to solve, either of the two methods may be more efficient.

The first method is to rewrite F to be of the form solved in Section 2.7. Let $g : 2^{\{1,\ldots,k\}} \rightarrow \mathcal{R}$ be such that

$$g(A) = f_i(X_{i,1}, \ldots, X_{i,k}) \text{ such that } X_{i,j} = \begin{cases} 1 & \text{if } j \in A \\ 0 & \text{if } j \notin A \end{cases}$$

(i.e., g of a set is f_i applied to its characteristic vector). The following lemma follows from the theory of harmonic analysis on the cube.

[1] In fact, we will require a uniformity condition, but will ignore this issue for now.

Lemma 3.1 (KKL) $g(A) = \sum_{S \subseteq \{1,\ldots,k\}} \alpha_S (-1)^{|S \cap A|}$, *where* $\alpha_S = 2^{-k} \sum_{B \subseteq \{1,\ldots,k\}} g(B)(-1)^{|S \cap B|}$.

Thus,

$$\begin{aligned} f_i(X_{i,1},\ldots,X_{i,k}) &= g(\{j \mid X_{i,j} = 1\}) \\ &= \sum_S \alpha_S (-1)^{|\{j \mid X_{i,j}=1\}|} \quad \text{(by Lemma 3.1)} \\ &= \sum_S \alpha_S (-1)^{\sum_{i \in S} X_{i,j}} \end{aligned}$$

Since we have now written F as $\sum_{i=1}^{n^{a+b}} \alpha_i (-1)^{\sum \cdot X_{i,j}}$, we can apply the technique of Section 2.7 to compute conditional expectations. This gives us the following theorem:

Theorem 3.2 *There is an NC algorithm which given any* $F : Z_2^n \to R$ *of the form*

$$F(X) = \sum_{i=1}^{n^a} f_i(X_{i,1},\ldots,X_{i,b\log n}),$$

outputs an X *with* $F(X) \geq E[F(X)]$.

An alternative method for computing conditional expectations for F is as follows. First note that, by linearity of expectation, it suffices to compute the conditional expectations of the individual f_i and sum. Assume we wish to compute $E[\, f_i(X_{i,1},\ldots,X_{i,b\log n}) \mid \omega_1 = s_1,\ldots,\omega_t = s_t\,]$. Let x be the vector $<X_{i,1},\ldots,X_{i,b\log n}>$, and let A be the matrix whose rows are the corresponding labels $a_{i,1},\ldots,a_{i,b\log n}$. Then $x = A\omega$. So

$$E[\, f_i(X_{i,1},\ldots,X_{i,b\log n}) \mid \omega_1 = s_1,\ldots,\omega_t = s_t\,] = \sum_x f_i(x) Pr[\, A\omega = x \mid \omega_1 = s_1,\ldots,\omega_t = s_t\,].$$

If we let $\omega' = <\omega_1,\ldots,\omega_t>$, $\omega'' = <\omega_{t+1},\ldots,\omega_l>$, A' and A'' be the first t and last $l - t$ columns of A respectively, and $s = <s_1,\ldots,s_t>$, then

$$\begin{aligned} E[\, f_i(x) \mid \omega_1 = s_1,\ldots,\omega_t = s_t\,] &= \sum_x f_i(x) Pr[\, A'\omega' + A''\omega'' = x \mid \omega' = s\,] \\ &= \sum_x f_i(x) Pr[\, A''\omega'' = x - A's\,]. \end{aligned}$$

For each x, we can test if the linear system $A''\omega'' = x - A's$ is solvable; if it is $Pr[\, A''\omega'' = x - A's\,] = 2^{\text{rank}(A'')+t-l}$, otherwise $Pr[\, A''\omega'' = x - A's\,] = 0$. Since we can compute the contribution of each of the x's in parallel, we can compute the desired conditional expectation in NC, thus giving an alternate proof for Theorem 3.2.

3.2 Polylogarithmic Number of 0/1 Variables

Now we will consider the case of functions depending on a polylogarithmic number of variables. A simple counting argument shows that in NC we cannot compute all functions of $\log^c n$ variables, when $c > 1$, let alone compute conditional expectations of them. In fact, both techniques of Section 3.1 require evaluating f_i at every point; if f_i depends on more than a logarithmic number of variables, there will be a superpolynomial number of points to evaluate. However, there are some special cases for which we can compute conditional expectations.

The first special case we can handle is

$$f_i(X_{i,1},\ldots,X_{i,k}) = (-1)^{\sum_j X_{i,j}}.$$

This function can be evaluated using the techniques of Section 2.7, even if $k = \log^c n$. In Section 3.1, we showed how to transform any function into a linear combination of these; if this transformation is already known and provides only a polynomial number of non-zero α's, we can use this technique.

The next special case will be based on the second technique of Section 3.1. Recall, we had

$$E[\, f_i(X_{i,1},\ldots,X_{i,k}) \mid \omega_1 = s_1,\ldots,\omega_t = s_t \,] = \sum_x f_i(x) Pr[\, A''\omega'' = x - A's \,].$$

We can restrict our attention to those x for which $f_i(x) \neq 0$. If there are a polynomial number of these, we can compute conditional expectations of f_i for $k = \log^c n$. Some examples of this are logical AND and NOR of a polylogarithmic number of variables (each has one non-zero point).

A variant of the above is the case of monomials, $f_i(X_{i,1},\ldots,X_{i,k}) = X_{i,1}X_{i,2}\cdots X_{i,k}$. This is equivalent to logical AND described above. Note that handling monomials is strictly weaker since, for example, it is impossible to write a poly-log variable NOR as a linear combination of a polynomial number of monomials.

Finally, we give a type of f_i which can simulate all the above and more. Consider functions of the form

$$f_i(x) = \begin{cases} 1 & \text{if } x = y + Tz \text{ for some } z \in Z_2^k \\ 0 & \text{otherwise} \end{cases}$$

for some $y \in Z_2^k$, $T \in Z_2^{k \prod k}$ (which may be different for different f_i).

These are the characteristic functions of affine subspaces. Included are characteristic functions of all single points; we can write any function with a polynomial number of non-zero points as a linear combination of these. Functions $(-1)^{\sum_j X_{i,j}}$ can also be put in this form. To compute conditional expectations, we use a variant of our linear algebra method:

$$\begin{aligned} E[\, f_i(x) \mid \omega_1 = s_1,\ldots,\omega_t = s_t \,] &= \sum_x f_i(x) Pr[\, A\omega = y + Tz \mid \omega' = s \,] \\ &= \sum_x f_i(x) Pr[\, A''\omega'' + Tz = y - A's \,] \end{aligned}$$

which can be computed by performing Gaussian Elimination to determine how many bits of ω'' are free to vary. This gives us the following theorem:

Theorem 3.3 *There is an NC algorithm which given any $F : Z_2^n \to R$ of the form*

$$F(X) = \sum_{i=1}^{n^a} f_i(X_{i,1},\ldots,X_{i,b\log^c n}),$$

where each f_i is the characteristic function of some affine subspace of $Z_2^{\log^c n}$, outputs an X with $F(X) \geq E[F(X)]$.

4 Handling Multivalues — the Hypergraph Coloring Problem

In the previous sections, we were only concerned with the case where the random variables took on values 0 and 1 each with probability 1/2. Yet, for many problems, this model is too restrictive.

In this section, we expand our framework to consider random variables drawn from a uniform distribution over a larger set of values. This can then be used to simulate non-uniform distributions. We will demonstrate our techniques for handling multivalued random variables on the following problem:

A *hypergraph* $\mathcal{H} = (V, \mathcal{E})$ is a system $\mathcal{E}$ of subsets of V called *edges*. $\mathcal{H}$ is d-uniform if every edge has d elements. Kleitman and Alon, Babai, and Itai [Kl, ABI] define the *large d-partite subhypergraph problem* as follows. Given a d-uniform hypergraph $\mathcal{H} = (V, \mathcal{E})$, find a d-coloring of V such that the number of edges in $\mathcal{E}$ having precisely one vertex of each color is at least $\left\lfloor |\mathcal{E}| d!/d^d \right\rfloor$. [ABI] showed this problem is in NC for constant d. We will show in this section that this problem is in NC for all d. Since the case of $d > \ln|\mathcal{E}| + \Omega(\lg\lg|\mathcal{E}|)$ is trivially satisfied by any coloring that colors one hyperedge correctly, we will henceforth restrict our attention to the case $d \leq \ln|\mathcal{E}| + O(\lg\lg|\mathcal{E}|)$.

4.1 Randomized Algorithm

In this section, we give a randomized parallel algorithm and prove that the expected number of properly colored edges is as desired. The randomized algorithm is as follows. Randomly assign to each vertex an $l = 3\log|\mathcal{E}|$ bit label. Designate these as random variables $Y = Y_1, \ldots, Y_{|V|}$. Let $\rho = \left\lfloor 2^l/l \right\rfloor$. A vertex is mapped to color i if its label is in $C_i = \{(i-1)\rho, \ldots, i\rho - 1\}$. Note that every color has ρ values associated with it. Vertices with values in the range $d\rho$ to $2^l - 1$ are uncolored. Note that fewer than d of the 2^l possible values yield an uncolored node.

Now for the analysis. We define a benefit function, $G(Y)$, which will be the sum of terms $g_e(Y)$, one for each $e \in \mathcal{E}$. Each $g_e(Y)$ will be 1 if the vertices of edge e are assigned d different colors, and 0 otherwise. In calculating the expected value of $g_e(Y)$, we get

$$
\begin{aligned}
& E[\,g_e(Y)\,] \\
&\geq Pr[\,g_e(Y) = 1 \mid \text{no vertex on edge } e \text{ gets non-color}\,] - Pr[\,\text{some vertex on edge } e \text{ gets non-color}\,] \\
&\geq \frac{d!}{d^d} - \frac{d^2}{2^l} \\
&= \frac{d!}{d^d} - \frac{d^2}{|\mathcal{E}|^3} \quad (\text{since } l = 3\log|\mathcal{E}|).
\end{aligned}
$$

Then,

$$
E[G(Y)] = \sum_{e\in\mathcal{E}} E[\,g_e(Y)\,] \geq |\mathcal{E}|\frac{d!}{d^d} - \frac{d^2}{|\mathcal{E}|^2} > \left\lfloor |\mathcal{E}|\frac{d!}{d^d} \right\rfloor - 1.
$$

In fact, since $G(Y)$ is integral, we know that $G(Y) \geq E[G(Y)]$ implies that $G(Y) \geq \left\lfloor |\mathcal{E}|\frac{d!}{d^d} \right\rfloor$, which is exactly what is desired.

4.2 The Basic Approach

In this section, we discuss various approaches for determinizing algorithms which use multivalued random variables. These approaches have different advantages and disadvantages and may all prove useful in applications.

The easiest approach to handling functions of multivalued random variables is to represent each variable by a collection of boolean random variables. In particular, for the large d-partite subhypergraph problem, if d is a power of 2, $d = O(\log|\mathcal{E}|/\log\log|\mathcal{E}|)$, we can represent the color of each vertex by $\lg d$ boolean random variables. Then each g_e would become a function of

$d \lg d = O(\log |\mathcal{E}|)$ boolean variables, allowing us to apply the general framework of Section 3 to find a good coloring.

A second approach we might consider would be to replace Z_2 in our distrubution with some other finite field GF(q). For the large d-partite subhypergraph problem, if d is any prime power, $d = O(\log |\mathcal{E}| / \log\log |\mathcal{E}|)$, we can replace Z_2 with GF(d). Theorem 2.8 will still hold and all of the approaches to get labels can be easily modified to work, giving us a distribution with $(\log n)$-wise independent random variables uniformily distributed over GF(d). Since d is small (we only require polynomial in n), we can try each possible value for the next element of ω in parallel and pick the one with the best conditional expected benefit G. To evaluate the conditional expectations, we can still use the linear algebra method of Section 3.1 to find the probability a collection of d random variables take on some particular value. And we can do this for each possible value, since d^d is polynomial in n. Thus we can still efficiently zero in on a good sample point.

To find a good coloring for any d up to $\log |\mathcal{E}|$, we must use a more complicated approach, one which is similar to the one used by Luby for $\Delta + 1$ vertex coloring [L2]. In essense, we repeatedly use the 0/1 problem as a subroutine to set one bit of the random variables at a time. We have multivalued random variables $Y = Y_1, \ldots, Y_{|V|}$ where $Y_i = Y_{i1} Y_{i2} \cdots Y_{il}$. We will compute the Y_i's bit by bit. At step t, we will compute $X^{(t)}$ such that

$$E[G(Y)|Y_{ij} = X_i^{(j)} \text{ for } 1 \le j \le t] \ge E[G(Y)|Y_{ij} = X_i^{(j)} \text{ for } 1 \le j \le t-1].$$

If we let

$$F^{(t)}(X^{(t)}) = E[G(Y)|Y_{ij} = X_i^{(j)} \text{ for } 1 \le j \le t],$$

then the above is equivalent to finding an $X^{(t)}$ with $F^{(t)}(X^{(t)}) \ge E[F^{(t)}(X^{(t)})]$. Letting

$$f_e^{(t)}(X^{(t)}) = E[g_e(Y)|Y_{ij} = X_i^{(j)} \text{ for } 1 \le j \le t]$$

allows us to write $F^{(t)}(X^{(t)})$ as a sum of $|\mathcal{E}|$ functions, each depending on at most $d \le \lg |\mathcal{E}|$ random variables $X_i^{(t)}$. Assuming that, given $X^{(1)}, \ldots, X^{(t-1)}$, we can construct functions $f_e^{(t)}$ (we will show how to do this in the next section) we can find a good $X^{(t)}$ using the general framework of Section 3.1.

A simple inductive argument shows that for all t,

$$E[G(Y)|Y_{ij} = X_i^{(j)} \text{ for } 1 \le j \le t] \ge E[G(Y)].$$

It follows that letting Y be such that $Y_{ij} = X_i^{(j)}$ for all i and j implies that $G(Y) \ge E[G(Y)]$.

4.3 The Deterministic Algorithm

To apply the last multivalued approach described in the previous section, we must show how to construct, for any t and for any settings of the first $t-1$ bits $X^{(1)}, \ldots, X^{(t-1)}$, functions

$$f_e^{(t)}(X^{(t)}) = E[g_e(Y)|Y_{ij} = X_i^{(j)} \text{ for } 1 \le j \le t].$$

To do this, we will show how to compute

$$E[g_e(Y)|Y_{ij} = X_i^{(j)} \text{ for } 1 \le j \le t];$$

it then suffices to plug in the given $X^{(1)}, \ldots, X^{(t-1)}$ and every possible setting of the variables $\{X_i^{(t)} | i \in e\}$ to construct $f_e^{(t)}$.

Given edge e and the first t bits of each label, $f_e^{(t)}$ will be the probability that the edge is properly colored. To calculate $f_e^{(t)}$, we sort the vertices of edge e into groups having the same t-bit prefix. For each t bit string α, we let S_α be the set of vertices which have prefix α and let I_α be the set of 2^{l-t} values which have prefix α. We let T_α be $\{1+\sum_{\beta<\alpha}|S_\beta|,\ldots,\sum_{\beta\leq\alpha}|S_\beta|\}$. Observe that edge e will be properly colored if and only if for each α the vertices in S_α are assigned the colors in T_α.

Now we can calculate $f_e^{(t)}$ as follows:

$$\begin{aligned} f_e^{(t)}(X^{(t)}) &= \prod_{|\alpha|=j+1} Pr[\text{ vertices in } S_\alpha \text{ are assigned colors in } T_\alpha\,] \\ &= \prod_{|\alpha|=j+1} |S_\alpha|! \prod_{i\in T_\alpha} Pr[\text{ vertex in } S_\alpha \text{ gets color } i\,] \\ &= \prod_{|\alpha|=j+1} |S_\alpha|! \prod_{i\in T_\alpha} \frac{|\,C_i\cap I_\alpha\,|}{2^{l-j-1}}. \end{aligned}$$

Theorem 4.1 *The large d-partite subhypergraph problem, finding a coloring of V which properly colors at least $\left\lfloor |\mathcal{E}|d!/d^d \right\rfloor$ edges, is in NC. In fact, a more careful analysis shows that finding a coloring of V which properly colors at least $|\mathcal{E}|d!/d^d$ edges is possible in NC.*

5 Remarks

There are many other problems for which discrepancy can be used to obtain a solution. It is interesting to observe that, although high independence is required for discrepancy (Remark 2.7), for other problems (e.g. set cover type problems) where it might appear one needs it [MNN2], pairwise independence suffices [BRS]. This makes a real difference, because with lower independence, it is possible to use far fewer processors, and even approach an optimal processor-time product.

As a final note, the discrepancy algorithm of Section 2 can be improved to yield a $2\sqrt{\Delta\log n}$ bound in the special case $\Delta=\log^c n$. The improved algorithm, besides achieving a better discrepancy bound, is a nice example of how to apply the techniques of this paper. The basic approach is as follows. For each $A\in\mathcal{A}$, we let $g_A(X)$ be 0 if $|\chi(A)|\leq 2\sqrt{\Delta\log n}$, and 1 otherwise. Let $G(X)=\sum_{A\in\mathcal{A}} g_A(X)$, i.e. G(X) is the number of unbalanced edges. We want to find an X such that $G(X)\leq E[G(X)]<1$. To do this we first partition Γ into a poly-log number of subsets $\Gamma_1,\ldots,\Gamma_r$, such that the intersection of any Γ_j with any $A\in\mathcal{A}$ is $O(\log\log n)$ (this can be done using $O(\log\log n)$-wise independence combined with some techniques from [BRS]). Then for each j in sequence, we can construct a function $F^{(j)}(X^{(j)})$, where $X^{(j)}$ are the random variables corresponding to Γ_j, which is the expected value of $G(X)$ conditioned upon the values of $X^{(1)},\ldots,X^{(j-1)}$ set already and the given $X^{(j)}$. Each $F^{(j)}$ will be a sum of functions depending on $O(\log\log n)$ variables each, so we can apply our general framework to find a good $X^{(j)}$. A simple inductive argument shows that when we are done, we have a good X. The number of processors required for this algorithm is $n\log^{O(1)} n$.

6 Acknowledgements

We thank Tom Leighton and David Shmoys for helpful discussions. We also thank David Shmoys for suggesting the parallel edge coloring problem, and Joel Spencer for suggesting the parallel set discrepancy problem.

References

[ABI] Alon, N., L. Babai, A. Itai, "A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem", *Journal of Algorithms*, 7, pp. 567-583, 1986.

[AIS] Awerbuch, B., A. Israeli, and Y. Shiloach, *Finding Euler Circuits in Logarithmic Parallel Time*, Proc. 16th Ann. ACM Symp. on Theory of Computing, 1984, pp. 249-257.

[B] Berger, B., "Data Structures for Removing Randomness", *MIT Lab. for Computer Science Technical Report*, MIT/LCS/TR-436, Dec. 13, 1988.

[BRS] Berger, B., J. Rompel, P. Shor, "Efficient NC Algorithms for Set Cover with Applications to Learning and Geometry", submitted to FOCS 1989.

[BS] Berger, B., P. Shor, personal communication.

[CGHFRS] Chor, B., O. Goldreich, J. Hastad, J. Friedman, S. Rudich, and R. Smolenski, *The Bit Extraction Problem or t-Resilient Functions*, Proc. 26th IEEE Symposium on Foundations of Computer Science, 1985, pp. 396-407.

[CH] Cole, R., J. Hopcroft, "On Edge Coloring Bipartite Graphs", *SIAM J. Comput.*, vol. 11, 1982, pp. 540-546.

[GK] Gabow, H.N., O. Kariv, "Algorithms for Edge Coloring Bipartite Graphs and Multigraphs", *SIAM J. Comput.*, vol. 11, 1982, pp. 117-129.

[KKL] Kahn, J., G. Kalai, N. Linial, *The Influence of Variables on Boolean Functions*, Proc. 29th IEEE Symposium on Foundations of Computer Science, 1988, pp. 68-80.

[K] Karloff, H., personal communication.

[KS] Karloff, H.J., D.B. Shmoys, "Efficient Parallel Algorithms for Edge Coloring Problems", *Journal of Algorithms*, 8, pp. 39-52, 1987.

[KW] Karp, R.M., A. Wigderson, "A Fast Parallel Algorithm for the Maximal Independent Set Problem", *JACM*, vol. 32, no. 4, October 1985, pp. 762-773.

[Kl] Kleitman, D., personal communication.

[LPV] Lev, G.F., N. Pippenger, and L.G. Valiant, "A Fast Parallel Algorithm for Routing in Permutation Networks", *IEEE Trans. Comput.*, vol. 30, 1981, pp. 93-100.

[L1] Luby, M., "A Simple Parallel Algorithm for the Maximal Independent Set Problem", *SIAM J. Comput.*, vol. 15, no. 4, November 1986, pp. 1036-1053.

[L2] Luby, M., *Removing Randomness in Parallel Computation Without a Processor Penalty*, Proc. 29th IEEE Symposium on Foundations of Computer Science, 1988, pp. 162-173.

[MS] MacWilliams, F.J., Sloane, N.J.A., *The Theory of Error Correcting Codes*, North-Holland, Amsterdam, 1977.

[MNN] Motwani, R., J. Naor, M. Naor, "A Generalized Technique for Derandomizing Parallel Algorithms ", *draft*, Jan. 17, 1989.

[MNN2] Motwani, R., J. Naor, M. Naor, personal communication.

[N] Naor, J., personal communication.

[Rab] Rabin, M.O., "Efficient Dispersal of Information for Security Load Balancing and Fault Tolerance", *JACM*, to appear.

[Rag] Raghavan, P., "Probabilistic Construction of Deterministic Algorithms: Approximating Packing Integer Programs", *JCSS*, vol. 37, no. 4, October 1988, pp. 130-143.

[S] Spencer, J., *Ten Lectures on the Probabilistic Method*, SIAM, Philadelphia, 1987.

[V] Vizing, V.G., "On an Estimate of the Chromatic Class of a P-graph", (in Russian), *Diskret. Anal.*, vol. 3, 1964, pp. 25-30.

OFFICIAL DISTRIBUTION LIST

Director Information Processing Techniques Office Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, VA 22209	2 copies
Office of Naval Research 800 North Quincy Street Arlington, VA 22217 Attn: Dr. R. Grafton, Code 433	2 copies
Director, Code 2627 Naval Research Laboratory Washington, DC 20375	6 copies
Defense Technical Information Center Cameron Station Alexandria, VA 22314	12 copies
National Science Foundation Office of Computing Activities 1800 G. Street, N.W. Washington, DC 20550 Attn: Program Director	2 copies
Dr. E.B. Royce, Code 38 Head, Research Department Naval Weapons Center China Lake, CA 93555	1 copy